天津大学

《计算机网络》课程设计报告

Web Server 的设计与实现

学　　号 3022244290

姓　　名 陈秋澄

学　　院 智能与计算学部

专　　业 计算机科学与技术

年　　级 2022

任课教师 赵增华老师

年　　月　　日

一、报告摘要

（简要介绍需要解决的具体问题、协议设计和实现，以及主要实验结果。）

二、任务需求分析（在第一周完成这部分内容）

2.1 文件架构分析

源文件主要由 include、samples、src、static_site 与 Makefile 五个文件夹组成，其中：

1. include 用来存放一些.h 文件。
2. samples 用于本地检验实验结果是否正确，采用了一些要求报文的样本。
3. src：储存 echo_serverecho_client 的源代码以及一些消息处理和解析界面(详细描述在功能模块的一部分)，其中 lexer.l、parser.y、parse.c 用于报文解析。
4. static_site 用于 GET 功能检验的静态浏览器网页。
5. Makefile 用于有效地编译、链接和运行整个 socket 项目。

2.2 实现 GET、HEAD 和 POST 三种基本方法

GET：请求指定的页面信息，并返回实体主体；

HEAD：类似于 GET 请求，但回复响应中没有具体内容，用于获取报纸头部信息；

POST：向指定资源提交数据处理请求（如提交表格或上传文件）。信息中包含数据。POST 请求可能导致新资源的建立和/或现有资源的修改。

2.3 每周任务分析说明

总的来说，本次实验要求我们能够实现正确分析客户端发送的请求包，并做出正确的响应；支持 HTTP 并行请求以及支持多个客户端并发访问。

1. 第一周需要实现简单 Echo Web Server，掌握 lex 和 yacc 在正确分析消息方法，实现服务器对客户端各类消息的正确响应。第一周的主要任务是让我们熟悉编程环境以及 socket 的基础功能。
2. 第二周需要在第一周的基础上进一步细化服务器端的响应，改进服务器的功能，使其能够正确响应 HTTP/1.1 请求消息，并按照 RFC 2616 实现中的定义 HEAD、GET 和 POST 的持久连接（persistent connection），具体要求如下：

(1) 若收到客户端发送的信息 GET, HEAD 和 POST 方法，服务器遵循 RFC2616

处理并响应该规定的信息。

(2) 支持 4 种 HTTP/1.1 错误代码：400，404，501，505 并且能够准确识别客户端信息并进行响应。

(3) 妥善管理接收缓冲区，避免客户请求消息过长导致缓冲区溢出，即处理 Pipelining 请求过长以及 CGI 处理 POST 请求时潜在的大量参数传递导致的溢出情景。

(4) 服务器可处理读写磁盘文件时遇到的错误(如不存在权限和文件、IO 错误等)。

(5) 创建简化的日志记录模块，记录格式化日志。

总的来讲，第二周的任务是优化 server 处理框架，实现独立的消息处理和日志模块等，解耦和、模块化的编程框架还为后续的编程提供了便利及较好的 可扩展性。

3. 服务器解析并发请求数的能力需要在第三周得到改进，具体要求如下：

(1)服务器可以连续响应客户端使用相同的服务器 TCP 同时发送多个请求 GET/HEAD/POST，即支持 HTTP pipelining。

(2)按服务器 RFC2616 规定的顺序处理 HTTP 并发请求。此外，对于 HTTP 如果服务器认为其中一个请求是错误的，并拒绝该请求，则服务器需要能够正确识别和分析并发的下一个请求。

4. 多个客户端的并发处理应在第四周实现，具体要求如下：

当服务器等待客户端发送下一个请求时，它可以同时处理来自其他客户端的请求，即服务器可以同时处理多个并发客户端。

将服务器能支持的最大连接设置为 1024(操作系统可用文件描述符数的最大值)。若一个客户端只发送了一半的请求，服务端应该继续为另一个客户端提供服务。

5. 选做任务 CGI

首先根据 RFC3875 以及 RFC2396 文档，实现 CGI 的请求。CGI 要求的处理主要取决于 URI 的不同，请求将用新的线程处理；这项任务的完成将标志着我们的服务器能够正确处理 POST 请求，并有能力作为后端程序与前端界面交互。虽然只是一个基础的界面，但至少通过我们的实现，我们将拥有非常基本的用户注册和登录接口的静态网页。

三、协议设计

3.1 总体设计（在第一周完成这部分内容）

任务主要由请求响应模块、日志模块、pipeline 处理模块和并发处理模块等四个模块组成，其中：

并发处理模块：采用 select()方法实现并发处理多个客户端的要求，即多个

server 可以支持多个连接，然后采用非阻塞的方式并发处理多个连接的要求，即每次顺序通过连接池，如果当前连接已准备就绪，则进行 recv 操作，并将接收到的信息交给 Pipeline 处理模块。该模块负责监控多个客户端的连接，接收和处理客户端的请求，并将请求交给相应的模块。

Pipeline 处理模块：实现 HTTP pipeline，逐一分析接收到的多个请求，并按接收到的顺序提交给请求响应模块。

请求响应模块：主要是对消息进行分析，并根据分析的信息进行响应；

日志模块：简化的日志记录模块，记录服务器的运行日志（包括请求信息、错误信息和其他关键操作等）。为了便于调试和跟踪，可以实现一个简化的日志记录模块来记录格式化的日志。格式是根据请求响应模块分析的信息和返回的响应代码将相关信息记录在日志文件中。

3.2 简单 echo web server 的设计

（从本节开始，下面每节对应一周的设计内容。每节都从数据结构设计、协议规则设计两个方面描述所做的设计，数据结构设计需要详细描述所设计协议的数据结构。包括协议头部结构、主要的数据结构等。协议规则设计需要详细描述协议完成各个功能的协议规则。）

3.2.1 数据结构设计

1. HTTP 协议报文：

在传输过程中，HTTP 报文中是 16 进制的 ASCII 码，这些十六进制的数字经过浏览器或者专用工具（如 wireshark）的翻译可以得到 HTTP 报文的结构。HTTP 的请求报文包括：请求行(request line)、请求头部(header)、空行和请求数据(request data) 四个部分组成。HTTP 的响应报文包括：状态行，响应头，空行，数据(响应体)。

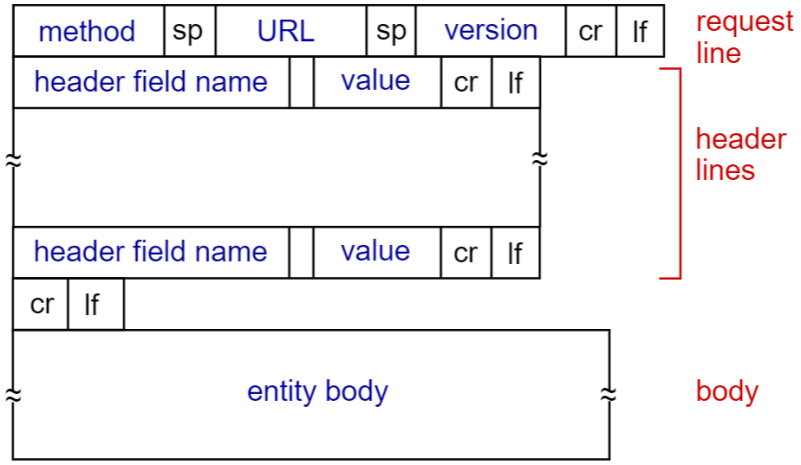

图 1　HTTP 请求报文的格式

2. 消息解析中的主要数据结构：
（1）Request_header：用于存储 HTTP 请求头的首部字段名和值。
（2）Request：用于存储 HTTP 请求消息的内容，包括：HTTP 版本号、请求方法、URL 以及相应的请求头和请求头的数量。
（3）sockaddr_in：用于记录 socket 的互联网地址，包括 IP 地址和端口号。
（4）addrinfo:用于保存 server 的相关信息，包括：输入 flag、协议族、socket 类型、socket 对应的协议、socket 地址长度、socket 地址等。
3. 响应报文的主要格式和结构：
HTTP 响应报告由三部分组成：状态行、响应头部和响应正文。
(1) 状态行:
由协议版本、状态码、状态码描述三部分组成，由空格分隔。
状态代码为 3 位数，其中 200-299 状态代码表示成功；300-399 状态代码指资源重定向；400-499 状态代码指客户端请求错误；500-599 状态代码指服务端错误（HTTP/1.1 的信息状态码在 100-199 的范围内引入到协议中)。
(2) 响应头部：
响应头部为响应报文添加了一些附加信息：

表 1：部分响应头部

名称	表示内容
Server	服务器应用程序的名称及版本
Content-Type	响应正文类型
Content-Length	响应正文长度
Content-Charset	响应正文所使用的编码
Content-Encoding	响应正文所使用的数据压缩格式
Content-Language	响应正文所使用的语种

(3) 响应正文：要读取的数据。
4. 判断文件状态的主要结构：fd_set

fd_set 是一组文件描述字的集合，它用一位来表示一个 fd 的状态是否可以读写或者访问。

3.2.2 协议规则设计

建立 HTTP 请求的完整过程： TCP 连接(之前可能有一次 DNS 域名解析)；三次握手建立 TCP 后，客户端向服务器发送请求命令，如 GET 等；客户端发送请求首部信息，然后发送空白行，GET 请求没有数据，POST 请求发送 body 数据；服务器收到上述信息后，开始处理业务，服务器开始响应；服务器返回响应头信息，发送 response header 之后，再发送一个空白行；然后服务器将数据发送到客户端；发送后，服务器四次挥手关闭 TCP 连接。
1. 请求消息的解析方法

从客户端发来的数据中获取请求信息，存储在 server 的缓存区中，调用 parse.c 中的 process 函数对消息进行解析，如果解析不出来说明格式有误返回对应提示，如果请求方式不是已知的三种给出无法实现的回应。
2. 接收缓冲区的设计

Server 的缓冲区已经定义好为 char buf[BUF_SIZE]作为接收缓冲区，我们又设置了一个 char *pbuf 作为发送缓冲区。同样在 echo_client.c 中有一个已设置

好的 char buf[BUF_SIZE]作为接收缓冲区。
第一周实验暂不涉及日志记录模块的设计。
3. 消息解析方法：

消息解析主要由功能模块中用来进行词法分析的 lex.yy.c、lexer.l、用来解析信息规则的定义与信息的 parse.y、parse.c 以及定义解析信息相关接口函数的 y.tab.c、y.tab.h 这三大模块来实现。首先使用 lex 进词法分析，按照 RFC 2016 中规定的数据类型来进行词法分析：将报文中的信息分为 cr、lf、sp、digit 等 16 个类型（完整代码见 lexer.l）。其中，parser.y 定义了解析由 lexer.l 定义的词法规则相关 token 的语法。例如以下的程序描述了一个关于请求头的语法定义，即一个请求开头，由 {"token"," 空格"," 文本"," 空格"," 文本"," 换行符"} 组成，且匹配到请求头时，要运行进行 request_line 有关代码。

接着由 yacc 来进行信息解析，yacc 通过 yyarse()（定义在 y.tab.c 中）来进行信息解析。首先根据 lex 分析定义的数据类型，然后根据 RFC 进行定义 2016 年 token、ows 和 text 的相应解析规则。然后按照 RFC 2016 定义请求行（request_line）和请求头（request_header）解析规则(详细代码在 parse 中)。此外，parse 函数通过调用 yyarse()函数实现对报文的解析，并存储在指定的 buff 中。

3.3 基本 HEAD、GET、POST 方法的设计

3.3.1 数据结构设计

3.3.2 协议规则设计

3.4 HTTP Pipelining 的设计

3.4.1 数据结构设计

3.4.2 协议规则设计

3.5 多个客户端的并发处理的设计

3.5.1 数据结构设计

3.5.2 协议规则设计

3.6 CGI 的设计（选做）

四、协议实现

（详细描述功能实现的细节。主要功能模块使用流程图或者伪代码来辅助说明。禁止贴源码。）

4.1 简单 echo web server 的实现

（从本节开始，下面每节对应一周的实现内容）

消息解析的具体实现如下：

(1)实现多头请求：

通过阅读 parse.y 代码可以知道，在定义请求（request）时：在规则中，每个请求只能与一个请求行和一个请求头部相匹配，这与 HTTP 报纸的格式有点不一致。在词法分析和分析信息时，yacc 和 lex 巧妙地运用递归，即递归定义变量 request_header_all 来匹配多个请求头部，也就是说 request_header_all 每次匹配时，都会匹配 request_header 或 request_header 和 request_header_all（通过这种方式实现了多头请求）。当然我们也可以修改 parser.y 文件中的规则，实现对多行请求的处理。具体为当匹配到‘token ows t_colon ows text ows t_crlf’的模式时，执行规则中代码，然后继续匹配下一个‘request_header’，可以用于处理多个连续的请求头的名称和值，将他们解析，注意对 request_header 的容量扩倍。

这里展示消息解析方法之 Lex 与 Yacc 的语法解析相关流程及原理图：

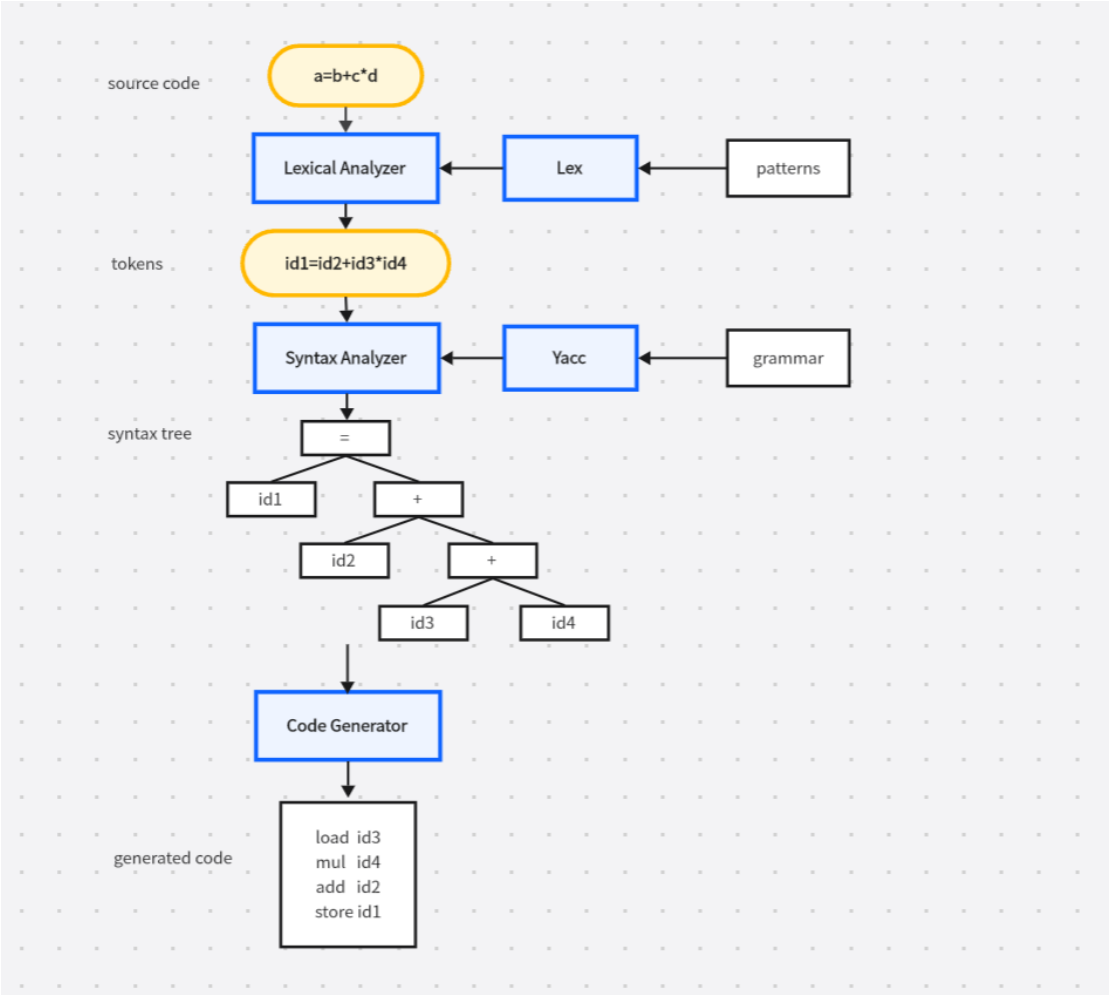

图 2　Lex 与 Yacc 语法解析相关流程及原理图

(2)扩大每个请求所能容纳的 header 数量：

通过阅读 parse.c 可以看出，每个为请求分配的请求头的数量为 1，考虑到多头请求的存在，注释掉下面的代码：

```
//request->headers = (Request_header *) malloc(sizeof(Request_header)*1);
```

通过在 parser.y 文件中 request_header:后面补充：

| request_header request_header{

YPRINTF("request_Header pattern 2\n");

};

即可使 request_header 每次匹配时，都会匹配 request_header 或 request_header，实现了多头请求。

(3)彻底清除缓冲区域：

在测试过程中，我发现在运行 request_get 示例后，我可以得到正确的答案；然后运行 request_400 示例，仍然可以得到正确的答案；最后，在再次运行 request_get 示例后，我得到了“HTTP/1.1 400 Bad request\r\n\r\n”的结果。

通过逐行打印，我发现 request 指针是 NULL。我发现在每次处理收到的信息后，当 request 和 buff 初始化时，buff 简单地对 buff 进行了 memset 处理和 request free 操作，这只是表面上的初始化，lex 和 yacc 在词法分析、消息解析中没有清空“看不见”的缓冲区和栈，通过进一步阅读源代码，我在 lex.yy.c 中找到了它 yylex_destroy()：在词法分析和消息解析中，该函数可以清空看不见的缓冲区和

栈。

(4)其他有关修改：

修改 echo_server.c 的输入参数个数为 4，包括一个测试文件路径。输入方式改为从 sample 文件中读取要发送的测试信息。

修改 echo_server.c，调用 parser.c 中的 char*process 函数对收到的信息解析，根据请求头和请求格式给出对应回应，然后通过 client_sock 进行回复。

这里，展示 Server 与 Client 设计时有关函数调用流程图：

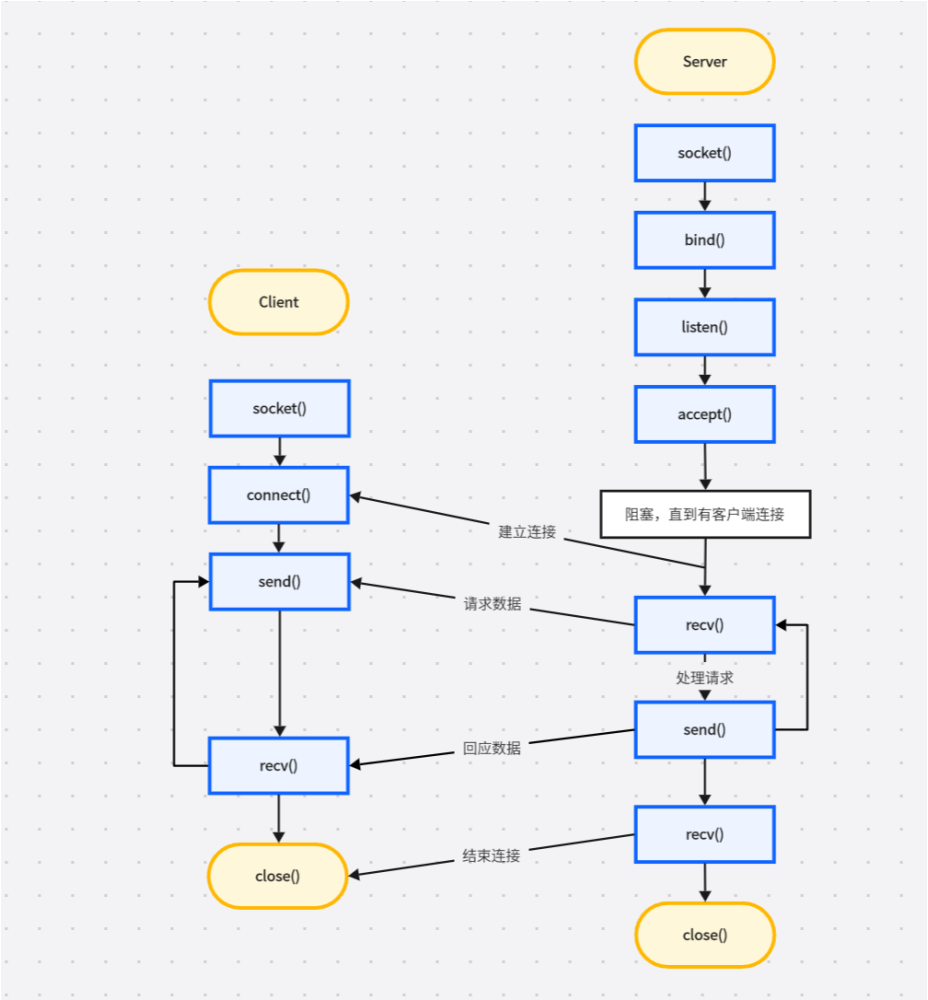

图 3　client 与 server 相关函数调用流程及原理

(5)关于 echo_server 接受消息并返回结果的实现：

在收到客户端 client 发送过来的消息之后，首先将其解析到 Request 类型的指针中，然后按照要求分成三种情况：

a) 收到的请求方法为 GET、HEAD 和 POST（即 Request 中的 http->method 为该三种方法中的一个），则直接返回收到的请求。

b) 出现格式错误问题：返回空指针，清空缓冲区，然后将“HTTP/1.1 400 Bad request\r\n\r\n”写入缓冲区，最后将消息传回给 client。

c) 收到非上述三种的其他请求方法：仍先清空缓冲区，再将“HTTP/1.1 501 Not Implemented\r\n\r\n”写入其中，然后传回信息。

下面，展示完成第一周后，服务器端收到报文的反应的流程：

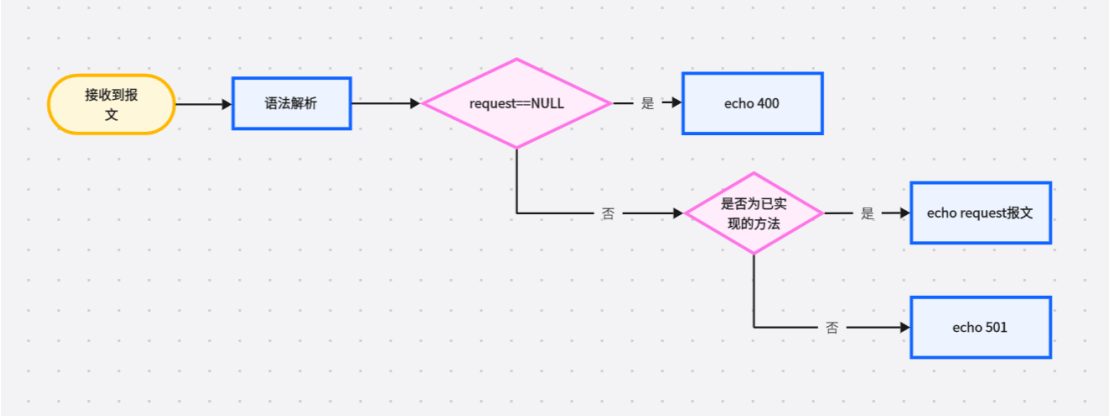

图 4 第一周实现后服务器端收到报文的反应

4.2 基本 HEAD、GET、POST 方法的实现

4.3 HTTP Pipelining 的实现

4.4 多个客户端的并发处理

4.5 CGI 的实现（选做）

五、实验结果及分析

（测试所实现协议的功能和性能，并对性能结果进行分析。需要针对考察点逐一展开。下面每节对应一周的内容）

5.1 简单 echo web server 的实验结果与分析

1. 本地手动测试结果
(1) GET、HEAD、POST
GET

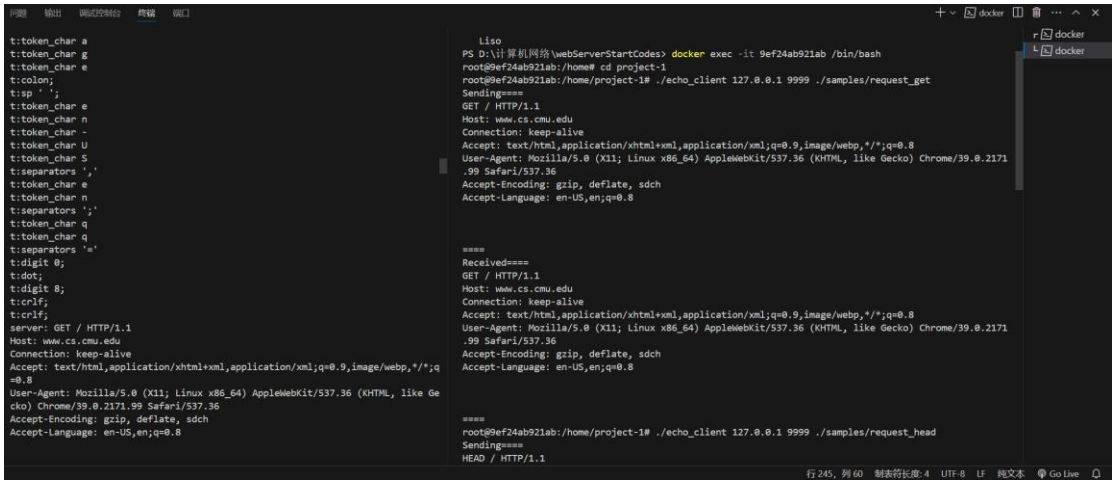

图 5 第一周 GET 测试截图

HEAD

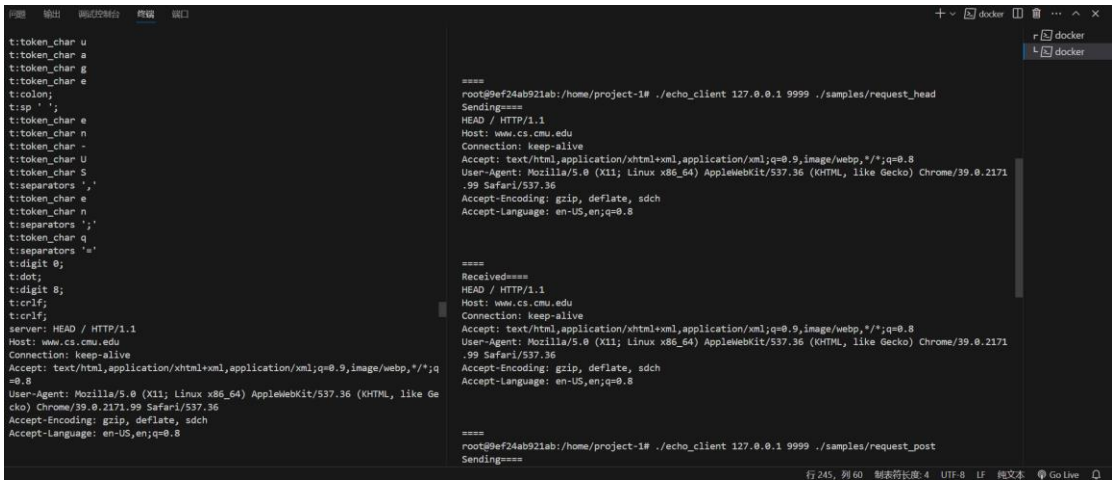

图 6 第一周 HEAD 测试截图

POST

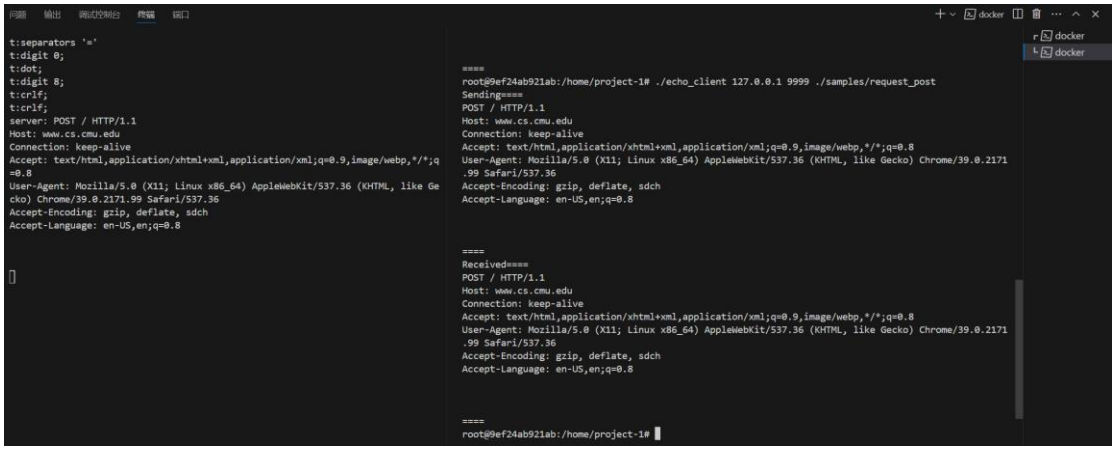

图 7 第一周 POST 测试截图

(2) 400 格式错误响应：

错误种类 1：多了换行

```
root@9ef24ab921ab:/home/project-1# ./echo_client 127.0.0.1 9999 ./samples/sample_400
Sending====
GET
/~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171
.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
====
Received====
HTTP/1.1 400 Bad request


====
root@9ef24ab921ab:/home/project-1#
```

图 8　400 格式错误响应 1

错误种类 2：开头了多空格

```
root@9ef24ab921ab:/home/project-1# ./echo_client 127.0.0.1 9999 ./samples/sample_error1
Sending====
    GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171
.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

====
Received====
HTTP/1.1 400 Bad request


====
root@9ef24ab921ab:/home/project-1#
```

图 9　400 格式错误响应 2

错误种类 3：报文中间多了字符

```
root@9ef24ab921ab:/home/project-1# ./echo_client 127.0.0.1 9999 ./samples/sample_error2
Sending====
GET /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171
.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language en-US,en;q=0.8

====
Received====
HTTP/1.1 400 Bad request


====
root@9ef24ab921ab:/home/project-1#
```

图 10　400 格式错误响应 3

错误种类 4：中间缺字符

```
====
root@9ef24ab921ab:/home/project-1# ./echo_client 127.0.0.1 9999 ./samples/sample_error3
Sending====
GET /~prs/15-441-F15/ HTTP/1.1
: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171
.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
====
Received====
HTTP/1.1 400 Bad request


====
root@9ef24ab921ab:/home/project-1#
```
行 7，列 32　空格: 4　UTF-8　CRLF　纯文本

图 11　400 格式错误响应 4

错误种类 5：无 HTTP/1.1 协议名称

```
====
root@9ef24ab921ab:/home/project-1# ./echo_client 127.0.0.1 9999 ./samples/sample_error4
Sending====
GET /~prs/15-441-F15/
Host: www.cs.cmu.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171
.99 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
====
Received====
HTTP/1.1 400 Bad request


====
```

图 12　400 格式错误响应 5

(3) 501 响应

```
root@9ef24ab921ab:/home/project-1# ./echo_client 127.0.0.1 9999 ./samples/sample_501
Sending====
GE /~prs/15-441-F15/ HTTP/1.1
Host: www.cs.cmu.edu



====
Received====
HTTP/1.1 501 Not Implemented


====
root@9ef24ab921ab:/home/project-1#
```

图 13　501 响应

2. 平台自动测试结果

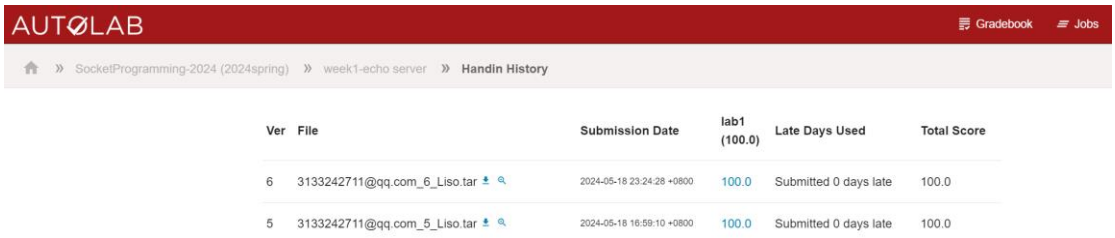

图 14 第一周平台测试 100 分结果

3. 实验结果的分析：

通过本地测试可知：能够对 GET、HEAD、POST 方法相应；能够对没有实现的方法进行对应的处理；能够识别五种格式错误并返回相应的信息。成功实现了简单的 echo web server。

5.2 HEAD、GET、POST 方法的实验结果与分析

5.3 HTTP 的并发请求的实验结果与分析

5.4 多个客户端的并发处理的实验结果与分析

六、总结

总结自己在实践过程中遇到的各类问题、困难以及解决过程中的收获，对实践内容等方面的体会与建议。

6.1 问题与解决

6.1.1 第一周

1. windows11 系统没有 Hyper-v，于是自行搜索，先创建.bat 文件，进行后续安装。

2. 出现报错：error during connect: this error may indicate that the docker daemon is not running

解决：重启 docker 程序，先在 docker 程序中进行登录，再在终端中进行容器开启连接等操作。

3. 出现报错：failed to solve: process "/bin/sh -c apt-get update && apt-get -y install gcc…did not complete successfully: exit code: 100

解决：在镜像下载时，起初我没有开启梯子，下载进度条不移动，几乎处于停滞状态，于是我开启梯子，下载可以正常进行；一段时间后，我在下载第二部分相关内容时关掉梯子，全部镜像文件得以成功下载完成。

4. 在运行原始代码时，我输入：

./echo_client 127.0.0.1 9999 ./samples/sample_request_example 时出现下图所示的

提示：

```
usage: ./echo_client <server-ip> <port>root@9ef24ab921ab:/home/project-1#
```

解决：阅读原始代码后，我发现正确的输入应为 3 项，应该先输入./echo_client 127.0.0.1 9999 后，再进行相关输入测试。

5. 一段时间未使用后，退出容器后重新尝试绑定但出现报错：

```
root@9ef24ab921ab:/home/project-1# ./echo_server
----- Echo Server -----
Failed binding socket.
root@9ef24ab921ab:/home/project-1#
```

图 15 报错截图

解决：重启电脑，server 成功绑定 socket。

6. 起初，无论我如何调整 Makefile 文件，压缩后提交至评测平台始终报错如下，得分为 0，但我在本地手动输入测试数据，结果均显示正确。

平台显示结果如下：

```
Compiling
make[1]: Entering directory '/home/autograde/autolab/Liso-handout'
rm -f  example echo_server echo_client src/lex.yy.c src/y.tab.*
rm -f -r obj
make[1]: Leaving directory '/home/autograde/autolab/Liso-handout'
make[1]: Entering directory '/home/autograde/autolab/Liso-handout'
gcc  -o example
gcc: fatal error: no input files
compilation terminated.
Makefile:22: recipe for target 'example' failed
make[1]: *** [example] Error 1
make[1]: Leaving directory '/home/autograde/autolab/Liso-handout'
Failure: Unable to compile (return status = 2)
{"scores": {"lab1": 0}}

Score for this problem: 0.0
```

图 16 第一周初始平台评测报错截图

仔细检查 Makefile 文件后发现未修改第 6 行代码。我将原始代码

```
OBJ_EXAMPLE := $(OBJ_DIR)/y.tab.o $(OBJ_DIR)/lex.yy.o $(OBJ_DIR)/parse.o $(OBJ_DIR)/example.o
```

的“_EXAMPLE”删除后，重新编译、压缩，提交到平台评测，得分终于变成 100 分。

Ver	File	Submission Date	lab1 (100.0)	Late Days Used	Total Score
5	3133242711@qq.com_5_Liso.tar	2024-05-18 16:59:10 +0800	100.0	Submitted 0 days late	100.0

图 17 第一周平台评测 100 分截图

6.2 收获、体会及建议